CSMA 2011
10e Colloque National en Calcul des Structures
9-13 mai 2011, Presqu'île de Giens (Var)

Une approche tensorielle pour la discrétisation EF d'EDP Implantation dans un environnement à objets

R. Saad[1], D. Eyheramendy[2]

[1] *LMA, CNRS UPR 7051, Marseille, France, saad@lma.cnrs-mrs.fr*
[2] *LMA, CNRS UPR 7051, Ecole Centrale Marseille, France, dominique.eyheramendy@centrale-marseille.fr*

Résumé — Les problèmes qui se posent aujourd'hui en mécanique numérique et des domaines liés sont complexes, et impliquent de plus en plus souvent plusieurs physiques à différentes échelles de temps et d'espace. Le traitement numérique d'un problème complexe est en général long et difficile. Il nécessite souvent la mise au point de solveurs d'équations aux dérivées partielles (méthodes de discrétisation. Dans cet article, nous présentons un cadre générique à objets pour la dérivation des formulations éléments finis. Il s'agit d'un environnement permettant le développement automatisé d'outils de simulation basés sur la dérivée de formes variationnelles ici dans un contexte éléments finis. L'approche proposée ici peut être étendue à d'autres types de résolution numérique. Les fondements de l'approche sont donnés.
Mots clefs — Eléments finis, orienté objet, analyse tensorielle, discrétisation automatique d'EDP.

1 Introduction

Jusque dans les années 90, les programmes éléments finis étaient constitués de plusieurs milliers de lignes de code généralement écrit en FORTRAN. Les problèmes d'ingénierie qui sont résolus aujourd'hui sont de plus en plus complexes. Il peut s'agir de traiter plusieurs physiques, et/ou plusieurs échelles d'espace et de temps ... Dans les années 90, l'application de la conception orientée objet s'est avérée être très efficace pour le développement de programmes flexibles dans le contexte de développements à grande échelle dans le contexte industriel. Parmi les pionniers, Fenves [1] a décrit les avantages de la programmation orientée-objet dans le cadre des éléments finis. Depuis cette origine, la programmation orientée objet a été largement appliqué à tous les domaines du calcul mécanique (voir par exemple Mackerle [2] et Eyheramendy et al. [3] et références incluses). Toutefois, ces codes traditionnels de calcul, y compris les outils industriels n'offrent pas toujours toute la flexibilité pour s'attaquer au défi des problèmes multi-physiques et multi-échelles qui se posent dans l'industrie et la recherche. Plus récemment, de nouvelles approches basées sur la calcul symbolique ont été proposées afin d'automatiser l'élaboration de modèles éléments finis, y compris pour la partie de discrétisation. Ces approches consistent généralement à construire automatiquement les matrices élémentaires et à les intégrer dans des codes plus classiques ou dans des bibliothèques d'éléments finis. Les exemples les plus avancés sont : Eyheramendy et al. [4], Logg & al. [5], Korelc [6]. Notez que dans Korelc [6], le modèle de comportement est également pris en considération au niveau symbolique. D'autre part, le code Comsol Multiphysics suit en quelque sorte une stratégie similaire, mais les algorithmes liés à la résolution éléments finis sont figés et donc imposés pour la résolution. Il est quasi indispensable d'avoir accès à tous les niveaux de l'algorithmique pour résoudre de nouveaux problèmes plus complexes.

Dans notre approche, nous proposons un nouvel environnement générique pour automatiser la dérivation de schémas de discrétisation d'équations aux dérivées partielles. Nous intégrons dans l'approche tous les aspects nécessaires pour le traitement de problèmes couplés en mécanique. L'originalité de la contribution réside dans l'utilisation d'un cadre d'analyse tensorielle. Celui-ci est intégré dans une approche à objets et représente la base de conception d'un environnement symbolique permettent de construire automatiquement les formes symboliques des contributions élémentaires au sein d'un problème complexe physique. Enfin la partie définissant les formes élémentaires de l'outil de simulation est crée automatiquement.

Nous présentons dans cet article le cadre de base orientée objet couvrant l'élaboration des contributions élémentaires dans le contexte de la discrétisation par éléments finis. Dans le paragraphe 2, nous proposons un rapide aperçu de la programmation orientée objet des éléments finis et les approches fondées sur le traitement symbolique des éléments finis développées jusqu'à aujourd'hui. Dans le paragraphe 2, nous présentons une approche mathématique générique de dérivation de modèle discret dans un cadre variationel. Dans le paragraphe 4, nous décrivons le cadre orienté objet intégrant ce cadre tensoriel dans un contexte des formes variationelles. Dans le paragraphe 5, l'approche est illustrée sur les équations de Navier-Stokes.

2 Sur les approches orientées objet et les approches symboliques en éléments finis

2.1 Les éléments finis en programmation orientée objet

Depuis les années 1990, l'application de la conception orientée objet aux éléments finis s'est avéré être très efficace pour le développement de programmes flexibles (voir Zimmermann [7] et Dubois-Pélerin [8], [9]). Depuis ces premières contributions, le paradigme orienté objet a été appliqué dans la plupart des domaines de la mécanique. Plus récemment, certains auteurs ont proposés des d'ajouter les concepts de structuration. Parmi ces auteurs Nikishkov [10], Eyheramendy [16] qui ont présenté des concepts de base de l'utilisation de Java comme une alternative pour la programmation des éléments finis et Mackie [11] et Heng [12] qui ont présenté des concepts similaires fondées sur une approche C#. Sans entrer dans les détails, quelques auteurs ont proposé différentes implémentations pour la mécanique en Java ou des domaines connexes: Vanderheyden [13] et Riley [14] en mécanique des fluides, Nikishkov [15] en mécanique et mécanique de la rupture, Eyheramendy [16] en calcul parallèle de dynamique de fluide, Eyheramendy [17] en mécanique des solides, Baudel [20] en électromagnétisme, Hauser [21] pour le calcul parallèle en mécanique. Notez que de nombreux auteurs ont préconisé l'utilisation de Java et ont comparé ses performances à d'autres langages de programmation (langage C): Nikishkov [22], Bull [23], Hauser [21], Eyheramendy [18].

2.2 Les approches symboliques en éléments finis

L'utilisation de logiciels de manipulation algébrique a toujours été un point d'intérêt pour les développements des éléments finis. Parmi les premiers travaux, on trouve ceux de Luft [24] et Noor [25], dans lequel sont décrits des méthodes pour générer automatiquement des matrices éléments finis. Par la suite, de nombreux travaux ont été présentés pour la résolution de différents types de problèmes par éléments finis utilisant des techniques de calculs symboliques : en électromagnétisme dans Yagawa [26] et pour la résolution des problèmes non linéaire par éléments finis (voir Eyheramendy et al. [4] et références incluses, et très récemment [19]). Plus récemment, Korelc [6] a présenté une approche en Mathematica permettant de générer automatiquement du code finie élément (formes élémentaires en y intégrant les aspects constitutifs). De même Logg & al. [5] ont présenté le projet FENICS dont l'objectif est de développer des modèles éléments finis et de générer grâce un précompilateur de formes variationnelles le code correspondant**.**

2 Une approche tensorielle pour la discrétisation par éléments finis

Soit $\{V_i\}_{i=1}^{r}$ un espace de fonctions, on considère les formes a et L défini sur le produit des espaces $V_1 \times V_2 \times \ldots \times V_r$ et on considère le problème variationnel standard :

$$\begin{cases} \text{Trouver } u_i \in V_i \text{ tel que } \forall w_j \in V_i \text{ on a} \\ a = \sum_{l=1}^{n} a_l(u_1, u_2, \ldots, u_r, w_{\alpha(l)}) = \sum_{k=1}^{m} L_k(w_{\beta(k)}) = L \end{cases}$$

Où n désigne le nombre de termes à gauche de la formulation, a_l désigne le $l^{\text{ème}}$ terme de la fonctionnelle a , $\alpha(l)$ la fonction test correspondant au $l^{\text{ème}}$ terme de la fonctionnelle a , m désigne le nombre de termes de la fonctionnelle L, L_k le $k^{\text{ème}}$ terme de la fonctionnelle L et $\beta(k)$ la fonction

test dans le $k^{\text{ème}}$ terme de la fonctionnelle L.
On peut ramener la forme a_l à une forme bilinéaire et la forme L_k à une forme linéaire (soit par linéarisation soit par choix du champ solution…) donc le problème devient

$$\begin{cases} \text{Trouver } u_{\beta(l)} \in V \text{ tel que } \forall w_{\alpha(l)} \in \hat{V} \text{ on a} \\ a_l^{u_i}(u_{\beta(l)}, w_{\alpha(l)}) = L_k^{w_j}(w_{\beta(k)}) \end{cases} \quad (1)$$

$a_l^{u_i} : \hat{V} \times V \to \mathbb{R}$ est une forme bilinéaire,
$L_k^{w_j} : \hat{V} \to \mathbb{R}$ est une forme linéaire
$(\hat{V}, V)$ une paire d'espaces des fonctions.
La méthode des éléments finis appliquée à l'équation (1) consiste à remplacer $(\hat{V}, V)$ par une paire d'espaces construits à partir de fonctions continues par morceau.
Considérons $\{\hat{\varphi}_j\}_{j=1}^M$ comme base d'espace de fonctions tests $\hat{V}$ et $\{\varphi_j\}_{j=1}^M$ comme base d'espace de fonctions solutions V, on peut exprimer la solution approximative U en utilisant la base de fonctions V, $U = \sum_{j=1}^M \xi_j \varphi_j$. Après dérivation, le problème se ramène à un système linéaire

$$A\xi = B$$

où $\{\xi_j\}_{j=1}^M$ sont les degrés de liberté correspondant à l'approximation U.
Les contributions au système sont calculées par assemblage des contributions élémentaires liées à chacun des termes de l'équation. On note que si la forme bilinéaire est exprimée comme une intégrale sur le domaine Ω, on peut écrire la forme $a_l^{u_i}$ comme somme de formes élémentaires :

$$a_l^{u_i,K}\left(\hat{\varphi}_j, \varphi_j\right) = \sum_{K \in \mathcal{D}} a_l^{u_i,K}\left(\hat{\varphi}_j, \varphi_j\right)$$

Où K est un élément de la discrétisation $\mathcal{D}$ de Ω.
Soit $\{\varphi_j^K\}_{j=1}^M$ et $\{\hat{\varphi}_j^K\}_{j=1}^M$ les restrictions respectives sur l'élément K des ensemble $\{\varphi_j\}_{j=1}^M$ et $\{\hat{\varphi}_j\}_{j=1}^M$. La forme $a_l^{u_i,K}$ définit un tenseur élémentaire d'ordre 2 de la forme:

$$A_\kappa^K = a_l^{u_i,K}\left(\varphi_j^K, \hat{\varphi}_j^K\right)$$

Où κ correspond à une combinaison entre les fonctions d'interpolations et les degrés de liberté.
Le tenseur élémentaire A_κ^K peut être calculé explicitement en le considérant comme une séquence de produit de tenseurs. Du point de vue pratique, dans la procédure de discrétisation, on présente chaque terme de la forme $a_l^{u_i,K}$ par un tenseur, puis on effectue les produits entre tenseurs correspondant, qui peut se mettre sous la forme :

$$A_\kappa^K = \int_K \prod_{j=1}^m D^j C_{\delta_j(\alpha,\beta)} \varphi_{\gamma_j(\alpha,\beta,\kappa_{(i,\beta)})}(x)\, dx \quad (2)$$

avec $i = (i_1, \dots, i_r)$ où r est l'ordre du tenseur A_κ^K et représente également le nombre de fonctions de base dans la discrétisation, m est le nombre de facteurs dans la forme $a_l^{u_i,K}$, D est l'operateur différentiel appliqué sur le champ, δ représente l'ensemble des indices du terme autre qu'un champ test ou un champ solution, κ correspond à une combinaison entre les fonctions d'interpolations et les degrés de liberté, α correspond aux degrés de liberté, β correspond au système de coordonnées de bases et γ représente l'ensemble des indices dans la fonction discrétisé. Cette forme est assez générale pour représenter tout type de formes variationnelles (forme multilinéaire ou forme non linéaire) dont les termes de base sont des champs tensoriels.

4 Une approche orientée objet pour le traitement symbolique des éléments finis

Dans ce paragraphe, nous décrivons les objets nécessaires aux manipulations symboliques pour la méthode des éléments finis. Tout d'abord, nous décrivons les classes nécessaires pour représenter tout type d'expression. Des classes supplémentaires sont nécessaires pour représenter les opérateurs

différentiels (appliqué aux tenseurs) et les dérivées partielles appliquées à une fonction. La classe Tensor est la représentation mathématique d'un tenseur. La méthode de discrétisation par éléments finis est programmée dans la classe **FEDiscretization.**

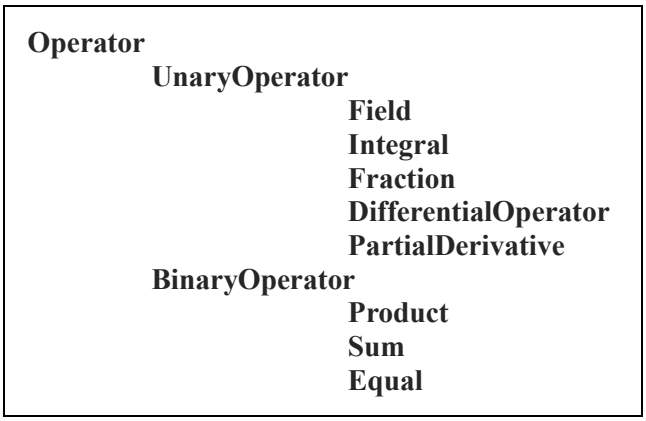

Figure 1: Une hiérarchie des operateurs

4.1 Classe Operator et ses sous-classes

La hiérarchie des classes qui est donné dans la figure 1 nous permet de représenter toutes sortes d'expressions. La classe **Field** représente un champ. Il pourrait être un scalaire, un vecteur ou un tenseur. Un opérateur aux dérivées partielles peut être appliqué à ce champ. Plus généralement, il peut être un tenseur. La classe **Integral** représente une intégrale. Elle est définie sur un domaine et possède un intégrant. Toutes les grandeurs numériques sont représentées dans la classe **Fraction**. Les expressions sont donc des sommes de produits gérées dans un arbre binaire représenté dans les classes **Sum** et **Product**. La classe **Equal** permet la représentation d'une égalité d'expressions, et donc d'une équation.

4.2 Classe Tensor

La classe **Tensor** est la représentation mathématique des tenseurs. Elle contient les composantes du tenseur qui sont des expressions (arbre binaire de champs). Les opérations tensorielles du base sont effectuées dans cette classe: produit tensoriel, produit contracté, somme…

4.3 Classe FEDiscretization

Cette classe regroupe toutes les opérations nécessaires à écrire le schéma de discrétisation par éléments finis. Ceci est fait pour les fonctions solutions et tests. Nous avons basé notre discrétisation sur la construction générique du tenseur A_{κ}^{K}.

4.4 Classe CodeCompiler

Cette classe gère l'intégration des formes élémentaires dans le code de calcul FEMJava (voir Eyheramendy [16, 17,18].

5 Application aux équations de Navier-Stokes

On considère les équations de Navier-Stokes pour un écoulement incompressible en 3D. On considère un élément cubique à huit nœuds. Une formulation faible stabilisée par ajout de termes de types moindres carrés du problème s'écrit de manière classique :

$$\begin{cases} \text{Trouver } (u,p) \in V; \text{tel que } \forall\ (v,q) \in \hat{V}, \text{on a} \\ \int_{\Omega} \rho \frac{\partial u}{\partial t} v\, d\Omega + \int_{\Omega} \rho(ugradu)v\, d\Omega - \int_{\Omega} 2\mu\varepsilon(u)\varepsilon(v)\, d\Omega + \int_{\Omega} pdivv\, d\Omega + \int_{\Omega} qdivu\, d\Omega - \int_{\Omega} fv\, d\Omega = 0 \end{cases}$$

Où u est la vitesse et p est la pression.

5.1 Approche tensorielle pour la discrétisation du terme de convection

A titre d'illustration, on considère le terme de convection qui correspond au second terme de la formulation:

$$a_2^{u_i,K} = \int_{\mathrm{K}} \rho(ugradu)v \, d\Omega$$

On identifie tous les paramètres pour calculer le tenseur élémentaire A_κ^K

$m = 4, r = 2,$

$$i = \begin{cases} i_1 = (N_1, \dots, N_8) \\ i_2 = (N_1, \dots, N_8) \end{cases}, \alpha = (x_1, x_2, x_3), \beta = (u_1, u_2, u_3), \kappa = \begin{cases} \kappa_1 = (i_1, \beta_1) \\ \kappa_2 = (i_2, \beta_2) \end{cases}$$

$$\gamma_j = \begin{cases} \gamma_1 = 0 \\ \gamma_2 = 0 \\ \gamma_3 = (\alpha_1, \beta_1, \kappa_1) \\ \gamma_4 = (\beta_1, \kappa_2) \end{cases}, \; D_j = \begin{cases} D^1 = 0 \\ D^2 = 0 \\ D^3 = \frac{\partial}{\partial \alpha} \\ D^4 = 0 \end{cases}, \delta_j = \begin{cases} \delta_1 = 0 \\ \delta_2 = \alpha_1 \\ \delta_3 = 0 \\ \delta_4 = 0 \end{cases}$$

Donc:

$$A_\kappa^K = \int_K \rho\big[E_{\alpha_1}\big]\big[F_{(\alpha_1,\beta_1,\kappa_1)}\big]\big[G_{(\beta_1,\kappa_2)}\big] \; dx$$

Les crochets représentent un tenseur provenant de la discrétisation du champ.
L'ensemble des indices est décrit comme suivant :

$\kappa_1 = (i_1, \beta_1) = \big((N_1, \dots, N_8), (u_1, u_2, u_3)\big)$ et $\kappa_2 = (i_2, \beta_2) = \big((N_1, \dots, N_8), (u_1, u_2, u_3)\big)$

On obtient le tenseur élémentaire

$$H_{\kappa_1\kappa_2} = \begin{pmatrix} \rho u_1 \frac{\partial \mathrm{N}_1}{\partial x_1}\mathrm{N}_1 & \rho u_1 \frac{\partial \mathrm{N}_1}{\partial x_2}\mathrm{N}_1 & \cdots & \rho u_1 \frac{\partial \mathrm{N}_1}{\partial x_1}\mathrm{N}_8 & \rho u_1 \frac{\partial \mathrm{N}_1}{\partial x_3}\mathrm{N}_8 \\ \rho u_2 \frac{\partial \mathrm{N}_1}{\partial x_1}\mathrm{N}_1 & \rho u_2 \frac{\partial \mathrm{N}_1}{\partial x_2}\mathrm{N}_1 & & \rho u_2 \frac{\partial \mathrm{N}_1}{\partial x_2}\mathrm{N}_8 & \rho u_2 \frac{\partial \mathrm{N}_1}{\partial x_2}\mathrm{N}_8 \\ \vdots & & \ddots & \vdots & \\ \rho u_2 \frac{\partial \mathrm{N}_8}{\partial x_1}\mathrm{N}_1 & \rho u_2 \frac{\partial \mathrm{N}_8}{\partial x_2}\mathrm{N}_1 & & \rho u_2 \frac{\partial \mathrm{N}_8}{\partial x_2}\mathrm{N}_8 & \rho u_2 \frac{\partial \mathrm{N}_8}{\partial x_3}\mathrm{N}_8 \\ \rho u_3 \frac{\partial \mathrm{N}_8}{\partial x_1}\mathrm{N}_1 & \rho u_3 \frac{\partial \mathrm{N}_8}{\partial x_2}\mathrm{N}_1 & \cdots & \rho u_3 \frac{\partial \mathrm{N}_8}{\partial x_2}\mathrm{N}_8 & \rho u_3 \frac{\partial \mathrm{N}_8}{\partial x_3}\mathrm{N}_8 \end{pmatrix}$$

La contribution élémentaire devient:

$$A_\kappa^K = \int_K H_{\kappa_1\kappa_2} dx$$

5.2 Traitement symbolique

Dans ce paragraphe, on considère les notations utilisées dans l'approche. Les notations sont évidentes et étroitement liée à la formulation mathématique classique. Nous commençons par la forme bilinéaire a:

```
a(u,v) = integral(rho u gradu v, D )
```

Les composantes du tenseur $grad(u)$ sont:

```
[1,1=[d(u_1)/dx_1];          1,2=[1/2(d(u_1)/dx_2+d(u_2)/dx_1)];

2,1=[1/2 (d(u_2)/dx_1 + d(u_1)/dx_2)];          2,2=[d(u_2)/dx_2]];
```

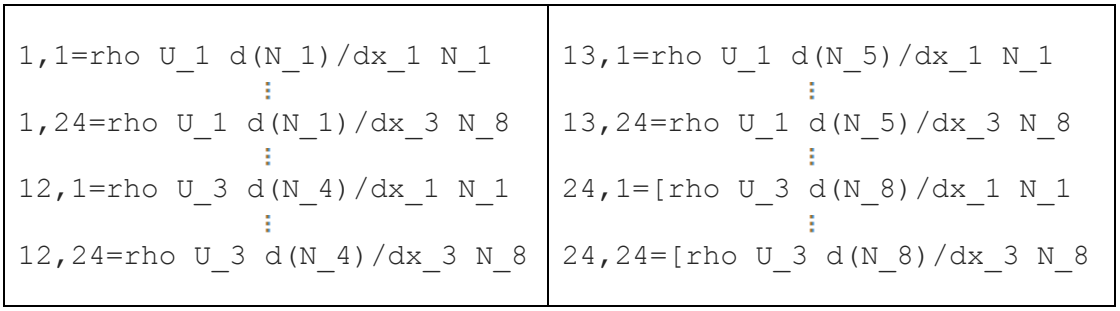

Figure 2: Le tenseur élémentaire correspondant au terme de convection de Navier-Stokes

6 Conclusion

Dans cet article, nous avons présenté la première étape d'un cadre global pour automatiser la génération de code éléments finis correspondant à un problème mécanique complexe. Les principes de discrétisation éléments finis automatisée basée sur une représentation tensorielle sont décrits. Le cadre présenté est structuré au sein d'un environnement orienté objet. D'autres types schémas de discrétisation pourraient être développés en utilisant une approche similaire. L'approche est illustrée sur l'exemple formulation variationnelle pour les équations de Navier-Stokes en incompressible (formulation stabilisée par ajout de termes de type moindres carrés). La formulation est intégrée à un code élément fini traditionnel écrit en Java [27].
Cette approche peut être étendue à un ensemble d'algorithmes numériques nécessaires pour obtenir un outil de simulation. Nous projetons d'étendre à court termes cette approche au cadre Lagrangien. Ce type d'approche vise à définir un concept à niveau d'abstraction élevé pour produire automatiquement un outil de calcul sans programmer une seule ligne de code. Le modèle orienté-objet présenté dans le présent document représente la méthode des éléments finis mathématiques qui peuvent être appliquées à tout type de problème. Cette approche générique ouvre de nouvelles voies dans la conception de codes basés sur le modèle mathématique et non plus sur de simples applications.

Références

[1] G.L. Fenves, *"Object-Oriented programming for engineering software development"*, Engineering with Computer, 6, 1-15, 1990.

[2] J. Mackerle, *"Object-oriented programming in FEM and BEM: a bibliography (1990-2003)",* Advances in Engineering Software, 35, 325-336, 2004.

[3] D. Eyheramendy, Th. Zimmermann, "*The Object-oriented finite elements: II. A symbolic environment for automatic programming",* Comput. Methods Appl. Mech. Engrg., 32, 259-276,.1996.

[4] D. Eyheramendy, Th. Zimmermann, *"Object-oriented finite elements: III. Theory and application of automatic programming"*, Comput. Methods Appl. Mech. Engrg., 154, 41- 68, 1998.

[5] A. Logg, *"Automating the Finite Element Method"*, Arch Comput Methods Eng., 14, 93-138, 2007.

[6] J. Korelc, *"Multi-language and Multi-environment Generation of Nonlinear Finite Element Codes"*, Engineering with Computers, 18, 312-327, 2002.

[7] Th. Zimmermann, Y. Dubois-Pèlerin and P. Bomme, *"Object-oriented finite element programming: I. Governing principles"*, Comput. Methods Appl. Mech. Engrg., 98, 291-303, 1992.

[8] Y. Dubois-Pèlerin, Th. Zimmermann and P. Bomme, *"Object-oriented finite element programming: II. A prototype program in Smalltalk"*, Comput. Methods Appl. Mech. Engrg., 98, 361-397, 1992.

[9] Y. Dubois-Pelerin and Th. Zimmermann, *"Object-oriented finite element programming: III – An efficient implementation in C++"*, Computer Methods Appl. Mech. Eng., 108(2), 165-183, 1993.

[10] G.P.Nikishkov, *"Object-oriented design of a finite element code in Java"*, Computer Modeling in Engng and Sciences, 11, 81-90, 2006.

[11] R.I. Mackie, *"Using Objects to handle complexity in Finite Element Software"*, Eng. with computers, 13, 99-111, 1997.

[12] B.C.P. Heng and R.I. Mackie, *"Design Patterns in Object-Oriented Finite Element Programming"*, in Proceedings of the Fifth International Conference on Engineering Computational Technology, B.H.V. Topping, G. Montero and R. Montenegro, (Editors), Civil-Comp Press, United Kingdom, 2006.

[13] W.B. VanderHeyden, E.D. Dendy and N.T. Padial-Collins, *"CartaBlanca-a pure-Java, component-based systems simulation tool for coupled nonlinear physics on unstructured grids-an update"*, Concurrency and Computation: Practice and Experience, 15, 431-458, 2003.

[14] C.J. Riley, S. Chatterjee and R. Biswas*, "High-performance Java codes for computational fluid dynamics"*, Concurrency and Computation: Practice and Experience 15, 395-415, 2003.

[15] G.P. Nikishkov and H. Kanda, *"The development of a Java engineering application for higher-order asymptotic analysis of crack-tip fields"*, Advances in Engineering Software, 30, 469-477, 1999.

[16] D. Eyheramendy*, "Object-oriented parallel CFD with JAVA"*, 15th International Conference on Parallel Computational Fluid Dynamics, Eds. Chetverushkin, Ecer, Satofuka, Périaux, Fox, Ed. Elsevier, 409-416, 2003.

[17] D. Eyheramendy and D. Guibert, "*A Java Approach for Finite Elements Computational Mechanics"*, ECCOMAS 2004, Jyvaskyla, Finland, July 2004.

[18] D. Eyheramendy and F. Oudin, Object-oriented finite elements: From Smalltalk to Java, In Trends in Engineering Computational Technology Eds. M. Papadrakakis and B.H.V. Topping, ©Saxe-Cobourg Publications, Chap. 2, (2008) pp. 17-40.

[19] R. Saad and D. Eyheramendy, An object-oriented framework for automated computer-aided finite element derivation, ECT 2010, 14-17 Sept. 2010, Valencia, Spain.

[20] L. Baduel, F. Baude, D. Caromel, C. Delbé, N. Gama, S. El Kasmi and S. Lanteri, *"A parallel object-oriented application for 3-D electromagnetism"*, ECCOMAS, Jyväskylä, Finland 2004.

[21] J. Häuser, T. Ludewig, R.D. Williams, R. Winkelmann, T. Gollnick, S. Brunett and J. Muylaert, "A test suite for high-performance parallel Java", Advances in Engineering Software, 31, 687-696, 2000.

[22] G.P. Nikishkov, Y.G Nikishkov and V.V Savchenko, *"Comparison of C and Java performance in finite element computations"*, Computer & Structures, 81, 2401-2408, 2000.

[23] J.M. Bull, L. A. Schmith, L. Pottage and R. Freeman, *"Benchmarking Java against C and Fortran for Scientific Applications"*, Joint ACM JavaGrande – ISCOPE 2001 Conference, Stanford Universtity, June 2-4, 2001.

[24] W. Luft, J.M. Roesset and J.J. Connor, *"Automatic generation of finite element matrices"*, Struct. Div. Proceedings of ASCE, 349-361, 1971.

[25] A.K. Noor, C.M. Andersen, *"Computerized symbolic manipulation in nonlinear finite element analysis"*, Computers & Structures, 13, 379-403, 1981.

[26] G. Yagawa, G.-W. Ye and S. Yoshimura, *"A numerical integration scheme for finite element method based on symbolic manipulation"*, International Journal for Numerical Methods in Engineering, 29, 1539-1549, 1990

[27] D. Eyheramendy, R. Saad, et F. Oudin *"Advanced object-oriented paradigms for parallel computational mechanics",* Proceedings of the 2nd International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering, B.H.V. Topping, J.M. Adam, F.J. Pallarés, R. Bru and M.L. Romero, (Editors), Civil-Comp Press, Stirlingshire, Scotland, 2011.